EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-
Naturwissenschaftliche
Fakultät
Wilhelm-Schickard-Institut

Typing, Representing, and Abstracting Control

Functional Pearl

Philipp Schuster, Jonathan Brachthäuser

- Implementation strategy for languages with control effects: CPS conversion

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  -

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  - Subtyping Delimited Continuations [Materzok and Biernacki 2011]

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  - Subtyping Delimited Continuations [Materzok and Biernacki 2011]
  - Representing Control [Danvy and Filinski 1992]

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  - Subtyping Delimited Continuations [Materzok and Biernacki 2011]
  - Representing Control [Danvy and Filinski 1992]
  - Abstracting Control [Danvy and Filinski 1990]

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  - Subtyping Delimited Continuations [Materzok and Biernacki 2011]
  - Representing Control [Danvy and Filinski 1992]
  - Abstracting Control [Danvy and Filinski 1990]
- Functional Pearl: parts fit perfectly

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  - Subtyping Delimited Continuations [Materzok and Biernacki 2011]
  - Representing Control [Danvy and Filinski 1992]
  - Abstracting Control [Danvy and Filinski 1990]
- Functional Pearl: parts fit perfectly
- Dependently typed (Idris): we index types of statements by a list of control effects

- Implementation strategy for languages with control effects: CPS conversion
- Based on:
  - Subtyping Delimited Continuations [Materzok and Biernacki 2011]
  - Representing Control [Danvy and Filinski 1992]
  - Abstracting Control [Danvy and Filinski 1990]
- Functional Pearl: parts fit perfectly
- Dependently typed (Idris): we index types of statements by a list of control effects
- Start with examples

Fail

$fail = \text{shift0}\ (\lambda k \Rightarrow \textbf{do}$
$\quad \text{pure}\ \underline{\texttt{"no"}})$

Fail

$$fail : \overline{\mathrm{Stm}}\ (\mathrm{String} :: rs)\ a$$

$$fail = \mathsf{shift0}\ (\lambda k \Rightarrow \textbf{do}$$

$$\quad \mathsf{pure}\ \texttt{"no"})$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Fail

$$fail : \overline{\mathrm{Stm}}\,(\mathrm{String} :: rs)\; a$$
$$fail = \mathsf{shift0}\;(\lambda k \Rightarrow \mathbf{do}$$
$$\quad \mathsf{pure}\;\texttt{"no"})$$

Flip

$$\mathit{flip} : \overline{\mathrm{Stm}}\ (r :: rs)\ \mathrm{Bool}$$
$$\mathit{flip} = \mathsf{shift0}\ (\lambda k \Rightarrow \textbf{do}$$
$$\quad \mathsf{resume}\ k\ \underline{\mathit{True}}$$
$$\quad \mathsf{resume}\ k\ \underline{\mathit{False}})$$

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Flip

$$\mathit{flip} : \overline{\mathrm{Stm}}\ (r :: rs)\ \mathrm{Bool}$$
$$\mathit{flip} = \mathsf{shift0}\ (\lambda k \Rightarrow \mathbf{do}$$
$$\quad \mathsf{resume}\ k\ \underline{\mathit{True}}$$
$$\quad \mathsf{resume}\ k\ \underline{\mathit{False}})$$

Emit

$$\mathit{emit} : \underline{a} \rightarrow \overline{\mathrm{Stm}}\ (\mathrm{List}\ a :: rs)\ ()$$
$$\mathit{emit}\ a = \mathsf{shift0}\ (\lambda k \Rightarrow \mathbf{do}$$
$$\quad as \leftarrow \mathsf{resume}\ k\ \underline{()}$$
$$\quad \mathsf{pure}\ (a\ \underline{::}\ as))$$

Emit

$$
\begin{aligned}
&\mathit{emit} : a \to \overline{\mathrm{Stm}}\ (\mathrm{List}\ a :: rs)\ ()\\
&\mathit{emit}\ a = \mathsf{shift0}\ (\lambda k \Rightarrow \mathbf{do}\\
&\quad as \leftarrow \mathsf{resume}\ k\ ()\\
&\quad \mathsf{pure}\,(a :: as)\,)
\end{aligned}
$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Emit Triples

$$emitTriples : \overline{\mathrm{Stm}}\ (\mathrm{String} :: \mathrm{List}\ (\mathrm{Int}, \mathrm{Int}, \mathrm{Int}) :: rs)\ \mathrm{String}$$

$$emitTriples = \mathbf{do}$$

$$\quad res \leftarrow triple\ \underline{9}\ \underline{15}$$

$$\quad \mathsf{lift}\ (emit\ res)$$

$$\quad \mathsf{pure}\ \underline{\texttt{"done"}}$$

$$emittedTriples : \overline{\mathrm{Stm}}\ [\,]\ (\mathrm{List}\ (\mathrm{Int}, \mathrm{Int}, \mathrm{Int}))$$

$$emittedTriples = \mathsf{reset0}\ (\mathsf{reset0}\ emitTriples \gg \mathsf{pure}\ \underline{[\,]})$$

Emit Triples

$$emitTriples : \overline{\text{Stm}}\ (\text{String} :: \text{List}\ (\text{Int}, \text{Int}, \text{Int}) :: rs)\ \text{String}$$

$emitTriples = \textbf{do}$

$\quad res \leftarrow triple\ \underline{9}\ \underline{15}$

$\quad \text{lift}\ (emit\ res)$

$\quad \text{pure}\ \underline{\texttt{"done"}}$

$$emittedTriples : \overline{\text{Stm}}\ [\,]\ (\text{List}\ (\text{Int}, \text{Int}, \text{Int}))$$

$$emittedTriples = \text{reset0}\ (\text{reset0}\ emitTriples \gg \text{pure}\ \underline{[\,]})$$

Emit Triples

$$\mathit{emitTriples} : \overline{\text{Stm}}\ (\text{String} :: \text{List}\ (\text{Int}, \text{Int}, \text{Int}) :: \mathit{rs})\ \text{String}$$

$$\begin{aligned}
&\mathit{emitTriples} = \mathbf{do} \\
&\quad \mathit{res} \leftarrow \mathit{triple}\ \underline{9}\ \underline{15} \\
&\quad \text{lift}\ (\mathit{emit}\ \mathit{res}) \\
&\quad \text{pure}\ \underline{\texttt{"done"}}
\end{aligned}$$

$$\mathit{emittedTriples} : \overline{\text{Stm}}\ [\,]\ (\text{List}\ (\text{Int}, \text{Int}, \text{Int}))$$

$$\mathit{emittedTriples} = \text{reset0}\ (\text{reset0}\ \mathit{emitTriples} \gg \text{pure}\ \underline{[\,]})$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Emit Triples

$$\mathit{emitTriples} : \overline{\mathrm{Stm}}\ (\mathrm{String} :: \mathrm{List}\ (\mathrm{Int}, \mathrm{Int}, \mathrm{Int}) :: \mathit{rs})\ \mathrm{String}$$

$$\mathit{emitTriples} = \mathbf{do}$$
$$\quad \mathit{res} \leftarrow \mathit{triple}\ \underline{9}\ \underline{15}$$
$$\quad \mathrm{lift}\ (\mathit{emit}\ \mathit{res})$$
$$\quad \mathrm{pure}\ \underline{\texttt{"done"}}$$

$$\mathit{emittedTriples} : \overline{\mathrm{Stm}}\ [\,]\ (\mathrm{List}\ (\mathrm{Int}, \mathrm{Int}, \mathrm{Int}))$$
$$\mathit{emittedTriples} = \mathrm{reset0}\ (\mathrm{reset0}\ \mathit{emitTriples} \gg \mathrm{pure}\ \underline{[\,]})$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Emit Triples

$$\mathit{emitTriples} : \overline{\mathrm{Stm}}\ (\mathrm{String} :: \mathrm{List}\ (\mathrm{Int}, \mathrm{Int}, \mathrm{Int}) :: \mathit{rs})\ \mathrm{String}$$
$$\mathit{emitTriples} = \textbf{do}$$
$$\quad \mathit{res} \leftarrow \mathit{triple}\ \underline{9}\ \underline{15}$$
$$\quad \mathsf{lift}\ (\mathit{emit}\ \mathit{res})$$
$$\quad \mathsf{pure}\ \underline{\texttt{"done"}}$$

$$\mathit{emittedTriples} : \overline{\mathrm{Stm}}\ [\,]\ (\mathrm{List}\ (\mathrm{Int}, \mathrm{Int}, \mathrm{Int}))$$
$$\mathit{emittedTriples} = \mathsf{reset0}\ (\mathsf{reset0}\ \mathit{emitTriples} \gg \mathsf{pure}\ \underline{[\,]})$$

Generated Code for Emitted Triples

$$
\begin{array}{l}
(\textbf{let } \mathit{f0}\ n = (\lambda k1 \Rightarrow (\lambda k2 \Rightarrow \\
\quad (\textbf{if } (n < 1) \\
\quad\quad \textbf{then } k2\ \texttt{"no"} \\
\quad\quad \textbf{else } \mathit{f0}\ (n - 1)\ k1\ (\lambda x4 \Rightarrow k1\ n\ k2)))) \textbf{ in } \mathit{f0})\ 9\ (\lambda x0 \Rightarrow (\lambda k3 \Rightarrow \\
(\textbf{let } \mathit{f2}\ n = (\lambda k1 \Rightarrow (\lambda k2 \Rightarrow \\
\quad (\textbf{if } (n < 1) \\
\quad\quad \textbf{then } k2\ \texttt{"no"} \\
\quad\quad \textbf{else } \mathit{f2}\ (n - 1)\ k1\ (\lambda x6 \Rightarrow k1\ n\ k2)))) \textbf{ in } \mathit{f2})\ (x0 - 1)\ (\lambda x1 \Rightarrow (\lambda k4 \Rightarrow \\
(\textbf{let } \mathit{f4}\ n = (\lambda k1 \Rightarrow (\lambda k2 \Rightarrow \\
\quad (\textbf{if } (n < 1) \\
\quad\quad \textbf{then } k2\ \texttt{"no"} \\
\quad\quad \textbf{else } \mathit{f4}\ (n - 1)\ k1\ (\lambda x8 \Rightarrow k1\ n\ k2)))) \textbf{ in } \mathit{f4})\ (x1 - 1)\ (\lambda x2 \Rightarrow (\lambda k5 \Rightarrow \\
\quad\quad\quad (\textbf{if } ((x0 + (x1 + x2)) \equiv 15) \\
\quad\quad\quad\quad \textbf{then } ((x0, x1, x2) :: (k5\ \texttt{"done"})) \\
\quad\quad\quad\quad \textbf{else } k5\ \texttt{"no"}))) \ k4))\ k3))\ (\lambda x0 \Rightarrow [\,])
\end{array}
$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Generated Code for Emitted Triples

$$
\begin{array}{l}
(\textbf{let}\ f0\ n = (\lambda k1 \Rightarrow (\lambda k2 \Rightarrow \\
\quad (\textbf{if}\ (n < 1) \\
\quad\quad \textbf{then}\ k2\ \texttt{"no"} \\
\quad\quad \textbf{else}\ f0\ (n-1)\ k1\ (\lambda x4 \Rightarrow k1\ n\ k2)))) \ \textbf{in}\ f0)\ 9\ (\lambda x0 \Rightarrow (\lambda k3 \Rightarrow \\
(\textbf{let}\ f2\ n = (\lambda k1 \Rightarrow (\lambda k2 \Rightarrow \\
\quad (\textbf{if}\ (n < 1) \\
\quad\quad \textbf{then}\ k2\ \texttt{"no"} \\
\quad\quad \textbf{else}\ f2\ (n-1)\ k1\ (\lambda x6 \Rightarrow k1\ n\ k2)))) \ \textbf{in}\ f2)\ (x0-1)\ (\lambda x1 \Rightarrow (\lambda k4 \Rightarrow \\
(\textbf{let}\ f4\ n = (\lambda k1 \Rightarrow (\lambda k2 \Rightarrow \\
\quad (\textbf{if}\ (n < 1) \\
\quad\quad \textbf{then}\ k2\ \texttt{"no"} \\
\quad\quad \textbf{else}\ f4\ (n-1)\ k1\ (\lambda x8 \Rightarrow k1\ n\ k2)))) \ \textbf{in}\ f4)\ (x1-1)\ (\lambda x2 \Rightarrow (\lambda k5 \Rightarrow \\
\quad\quad\quad (\textbf{if}\ ((x0 + (x1 + x2)) \equiv 15) \\
\quad\quad\quad\quad \textbf{then}\ ((x0, x1, x2) :: (k5\ \texttt{"done"})) \\
\quad\quad\quad\quad \textbf{else}\ k5\ \texttt{"no"})))\ k4))\ k3))\ (\lambda x0 \Rightarrow [\,])
\end{array}
$$

Generated Code for Emitted Triples

$$
\begin{array}{l}
(\textbf{let } \mathit{f0}\ n = (\lambda \mathit{k1} \Rightarrow (\lambda \mathit{k2} \Rightarrow \\
\quad (\textbf{if } (n < 1) \\
\quad\quad \textbf{then } \mathit{k2}\ \texttt{"no"} \\
\quad\quad \textbf{else } \mathit{f0}\ (n - 1)\ \mathit{k1}\ (\lambda \mathit{x4} \Rightarrow \mathit{k1}\ n\ \mathit{k2})))) \textbf{ in } \mathit{f0})\ 9\ (\lambda \mathit{x0} \Rightarrow (\lambda \mathit{k3} \Rightarrow \\
(\textbf{let } \mathit{f2}\ n = (\lambda \mathit{k1} \Rightarrow (\lambda \mathit{k2} \Rightarrow \\
\quad (\textbf{if } (n < 1) \\
\quad\quad \textbf{then } \mathit{k2}\ \texttt{"no"} \\
\quad\quad \textbf{else } \mathit{f2}\ (n - 1)\ \mathit{k1}\ (\lambda \mathit{x6} \Rightarrow \mathit{k1}\ n\ \mathit{k2})))) \textbf{ in } \mathit{f2})\ (\mathit{x0} - 1)\ (\lambda \mathit{x1} \Rightarrow (\lambda \mathit{k4} \Rightarrow \\
(\textbf{let } \mathit{f4}\ n = (\lambda \mathit{k1} \Rightarrow (\lambda \mathit{k2} \Rightarrow \\
\quad (\textbf{if } (n < 1) \\
\quad\quad \textbf{then } \mathit{k2}\ \texttt{"no"} \\
\quad\quad \textbf{else } \mathit{f4}\ (n - 1)\ \mathit{k1}\ (\lambda \mathit{x8} \Rightarrow \mathit{k1}\ n\ \mathit{k2})))) \textbf{ in } \mathit{f4})\ (\mathit{x1} - 1)\ (\lambda \mathit{x2} \Rightarrow (\lambda \mathit{k5} \Rightarrow \\
\quad\quad\quad (\textbf{if } ((\mathit{x0} + (\mathit{x1} + \mathit{x2})) \equiv 15) \\
\quad\quad\quad\quad \textbf{then } ((\mathit{x0}, \mathit{x1}, \mathit{x2}) :: (\mathit{k5}\ \texttt{"done"})) \\
\quad\quad\quad\quad \textbf{else } \mathit{k5}\ \texttt{"no"}))) \ \mathit{k4}))\ \mathit{k3}))\ (\lambda \mathit{x0} \Rightarrow [\,])
\end{array}
$$

Typing, Representing, and Abstracting Control

Basics: Continuation Passing Style

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Basics: Continuation Passing Style

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$

Basics: Continuation Passing Style

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

Basics: Continuation Passing Style

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Cps}\ r\ a \to (a \to r) \to r$$

Basics: Continuation Passing Style

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Cps}\ r\ a \to (a \to r) \to r$$
$$\mathsf{run0} = \mathsf{id}$$

Basics: Continuation Passing Style

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Cps}\ r\ a \to (a \to r) \to r$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \mathrm{Cps}\ r\ r \to r$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{id}$$

Basics: Continuation Monad

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

Basics: Continuation Monad

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{pure} : a \to \mathrm{Cps}\ r\ a$$
$$\mathsf{pure}\ a = \lambda k \Rightarrow k\ a$$

Basics: Continuation Monad

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\text{pure} : a \to \mathrm{Cps}\ r\ a$$
$$\text{pure}\ a = \lambda k \Rightarrow k\ a$$

$$\text{push} : (a \to \mathrm{Cps}\ r\ b) \to (b \to r) \to (a \to r)$$
$$\text{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

Basics: Continuation Monad

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{pure} : a \to \mathrm{Cps}\ r\ a$$
$$\mathsf{pure}\ a = \lambda k \Rightarrow k\ a$$

$$\mathsf{push} : (a \to \mathrm{Cps}\ r\ b) \to (b \to r) \to (a \to r)$$
$$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\mathsf{bind} : \mathrm{Cps}\ r\ a \to (a \to \mathrm{Cps}\ r\ b) \to \mathrm{Cps}\ r\ b$$
$$\mathsf{bind}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$$

Typing, *Representing*, and Abstracting Control

Representing Control: Target Language

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Exp} : \mathrm{Type} \to \mathrm{Type}\ \ \textbf{where}\\
&\quad \mathsf{Lam} : (\mathsf{Exp}\ a \to \mathsf{Exp}\ b) \to \mathsf{Exp}\ (a \to b)\\
&\quad \mathsf{App} : \mathsf{Exp}\ (a \to b) \to \mathsf{Exp}\ a \to \mathsf{Exp}\ b\\
&\quad \mathsf{Add} : \mathsf{Exp}\ \mathrm{Int} \to \mathsf{Exp}\ \mathrm{Int} \to \mathsf{Exp}\ \mathrm{Int}\\
&\quad \mathsf{Lit0} : \mathsf{Exp}\ \mathrm{Int}\\
&\quad \mathsf{Lit1} : \mathsf{Exp}\ \mathrm{Int}\\
&\quad \ldots
\end{aligned}
$$

Representing Control: Target Language

$$\textbf{data}\ \underline{\cdot} : \text{Type} \to \text{Type}\ \ \textbf{where}$$
$$\underline{\lambda} : (\underline{a} \to \underline{b}) \to \underline{a \to b}$$
$$\underline{@} : \underline{a \to b} \to \underline{a} \to \underline{b}$$
$$\underline{+} : \underline{\text{Int}} \to \underline{\text{Int}} \to \underline{\text{Int}}$$
$$\underline{0} : \underline{\text{Int}}$$
$$\underline{1} : \underline{\text{Int}}$$
$$\ldots$$

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Representing Control: Operators (before staging)

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Cps}\ r\ a \to (a \to r) \to r$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \mathrm{Cps}\ r\ r \to r$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{id}$$

Representing Control: Operators (after staging)

$$\overline{\mathrm{Cps}} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\overline{\mathrm{Cps}}\ r\ a = (\underline{a} \to \underline{r}) \to \underline{r}$$

$$\mathsf{shift0} : ((\underline{a} \to \underline{r}) \to \underline{r}) \to \overline{\mathrm{Cps}}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \overline{\mathrm{Cps}}\ r\ a \to (\underline{a} \to \underline{r}) \to \underline{r}$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \overline{\mathrm{Cps}}\ r\ r \to \underline{r}$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{id}$$

Representing Control: Monad (before staging)

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{pure} : a \to \mathrm{Cps}\ r\ a$$
$$\mathsf{pure}\ a = \lambda k \Rightarrow k\ a$$

$$\mathsf{push} : (a \to \mathrm{Cps}\ r\ b) \to (b \to r) \to (a \to r)$$
$$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\mathsf{bind} : \mathrm{Cps}\ r\ a \to (a \to \mathrm{Cps}\ r\ b) \to \mathrm{Cps}\ r\ b$$
$$\mathsf{bind}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Representing Control: Monad (after staging)

$$\overline{\text{Cps}} : \text{Type} \to \text{Type} \to \text{Type}$$
$$\overline{\text{Cps}}\ r\ a = (\underline{a} \to \underline{r}) \to \underline{r}$$

$$\text{pure} : \underline{a} \to \overline{\text{Cps}}\ r\ a$$
$$\text{pure}\ a = \lambda k \Rightarrow k\ a$$

$$\text{push} : (\underline{a} \to \overline{\text{Cps}}\ r\ b) \to (\underline{b} \to \underline{r}) \to (\underline{a} \to \underline{r})$$
$$\text{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\text{bind} : \overline{\text{Cps}}\ r\ a \to (\underline{a} \to \overline{\text{Cps}}\ r\ b) \to \overline{\text{Cps}}\ r\ b$$
$$\text{bind}\ m\ f = \lambda k \Rightarrow m\ (\text{push}\ f\ k)$$

Representing Control: Reify and Reflect

$$\text{reify} : \overline{\mathrm{Cps}}\ r\ a \to \underline{\mathrm{Cps}\ r\ a}$$

$$\text{reflect} : \underline{\mathrm{Cps}\ r\ a} \to \overline{\mathrm{Cps}}\ r\ a$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Representing Control: Reify and Reflect

$\text{reify} : \overline{\mathrm{Cps}}\ r\ a \rightarrow \underline{\mathrm{Cps}\ r\ a}$

$\text{reify}\ m = \underline{\lambda}\ \lambda k \Rightarrow m\ (\lambda a \Rightarrow k\ \underline{@}\ a)$

$\text{reflect} : \underline{\mathrm{Cps}\ r\ a} \rightarrow \overline{\mathrm{Cps}}\ r\ a$

$\text{reflect}\ m = \lambda k \Rightarrow m\ \underline{@}\ (\underline{\lambda}\ \lambda a \Rightarrow k\ a)$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Typing, Representing, and *Abstracting* Control

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Abstracting Control: CPS Hierarchy

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathrm{Cps}\ r\ a$$

Abstracting Control: CPS Hierarchy

$$\text{Cps} : \text{Type} \to \text{Type} \to \text{Type}$$
$$\text{Cps}\ r\ a = (a \to r) \to r$$

$$\text{Cps}\ r\ a$$
$$\text{Cps}\ (\text{Cps}\ r\ q)\ a$$

Abstracting Control: CPS Hierarchy

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathrm{Cps}\ r\ a$$

$$\mathrm{Cps}\ (\mathrm{Cps}\ r\ q)\ a$$

$$\mathrm{Cps}\ (\mathrm{Cps}\ (\mathrm{Cps}\ r\ q)\ p)\ a$$

$$\ldots$$

Abstracting Control: Effectful Statements

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathrm{Stm} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Stm}\ []\ a = a$$
$$\mathrm{Stm}\ (r :: rs)\ a = \mathrm{Cps}\ (\mathrm{Stm}\ rs\ r)\ a$$

$$\mathrm{Stm}\ [p, q, r]\ a = \mathrm{Cps}\ (\mathrm{Cps}\ (\mathrm{Cps}\ r\ q)\ p)\ a$$

Abstracting Control: Operators (before abstraction)

$$\mathrm{Cps} : \mathrm{Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Cps}\ r\ a = (a \to r) \to r$$

$$\mathsf{shift0} : ((a \to r) \to r) \to \mathrm{Cps}\ r\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Cps}\ r\ a \to (a \to r) \to r$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \mathrm{Cps}\ r\ r \to r$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{id}$$

Abstracting Control: Operators (after abstraction)

$$\mathrm{Stm} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Stm}\ [\,]\ a = a$$
$$\mathrm{Stm}\ (r :: rs)\ a = \mathrm{Cps}\ (\mathrm{Stm}\ rs\ r)\ a$$

$$\mathsf{shift0} : ((a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ (r :: rs)\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Stm}\ (r :: rs)\ a \to (a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \mathrm{Stm}\ (a :: rs)\ a \to \mathrm{Stm}\ rs\ a$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{pure}$$

Abstracting Control: Monad (before abstraction)

$\mathrm{Cps} : \mathrm{Type} \rightarrow \mathrm{Type} \rightarrow \mathrm{Type}$
$\mathrm{Cps}\ r\ a = (a \rightarrow r) \rightarrow r$

$\mathsf{pure} : a \rightarrow \mathrm{Cps}\ r\ a$
$\mathsf{pure}\ a = \lambda k \Rightarrow k\ a$

$\mathsf{push} : (a \rightarrow \mathrm{Cps}\ r\ b) \rightarrow (b \rightarrow r) \rightarrow (a \rightarrow r)$
$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$

$\mathsf{bind} : \mathrm{Cps}\ r\ a \rightarrow (a \rightarrow \mathrm{Cps}\ r\ b) \rightarrow \mathrm{Cps}\ r\ b$
$\mathsf{bind}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$

Abstracting Control: Monad (after abstraction)

$$\mathrm{Stm} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Stm}\ []\ a = a$$
$$\mathrm{Stm}\ (r :: rs)\ a = \mathrm{Cps}\ (\mathrm{Stm}\ rs\ r)\ a$$

$$\mathsf{pure} : a \to \mathrm{Stm}\ rs\ a$$
$$\mathsf{pure}_{r::rs}\ a = \lambda k \Rightarrow k\ a$$

$$\mathsf{push} : (a \to \mathrm{Stm}\ (r :: rs)\ b) \to (b \to \mathrm{Stm}\ rs\ r) \to (a \to \mathrm{Stm}\ rs\ r)$$
$$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\mathsf{bind} : \mathrm{Stm}\ rs\ a \to (a \to \mathrm{Stm}\ rs\ b) \to \mathrm{Stm}\ rs\ b$$
$$\mathsf{bind}_{r::rs}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$$

Abstracting Control: Monad (after abstraction)

$$\mathrm{Stm} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Stm}\ []\ a = a$$
$$\mathrm{Stm}\ (r :: rs)\ a = \mathrm{Cps}\ (\mathrm{Stm}\ rs\ r)\ a$$

$$\mathsf{pure} : a \to \mathrm{Stm}\ rs\ a$$
$$\mathsf{pure}_{r::rs}\ a = \lambda k \Rightarrow k\ a$$
$$\mathsf{pure}_{[]}\quad a = a$$

$$\mathsf{push} : (a \to \mathrm{Stm}\ (r :: rs)\ b) \to (b \to \mathrm{Stm}\ rs\ r) \to (a \to \mathrm{Stm}\ rs\ r)$$
$$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\mathsf{bind} : \mathrm{Stm}\ rs\ a \to (a \to \mathrm{Stm}\ rs\ b) \to \mathrm{Stm}\ rs\ b$$
$$\mathsf{bind}_{r::rs}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$$
$$\mathsf{bind}_{[]}\quad m\ f = f\ m$$

Abstracting Control: Lifting

$$\text{lift} : \mathrm{Stm}\ rs\ a \to \mathrm{Stm}\ (r :: rs)\ a$$
$$\text{lift} = \text{bind}$$

Abstracting Control: Lifting

$$\text{lift} : \mathrm{Stm}\ rs\ a \to \mathrm{Stm}\ (r :: rs)\ a$$
$$\text{lift} = \text{bind}$$

$$\text{shift0}_0 : ((a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ (r :: rs)\ a$$
$$\text{shift0}_0 = \text{shift0}$$

$$\text{shift0}_1 : ((a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ (q :: r :: rs)\ a$$
$$\text{shift0}_1 = \text{lift} \circ \text{shift0}$$

$$\text{shift0}_2 : ((a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ (p :: q :: r :: rs)\ a$$
$$\text{shift0}_2 = \text{lift} \circ \text{lift} \circ \text{shift0}$$

...

Typing, *Representing*, and *Abstracting* Control

Representing Abstracted Control: Operators

$$\mathrm{Stm} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Stm}\ []\ a = a$$
$$\mathrm{Stm}\ (r :: rs)\ a = \mathrm{Cps}\ (\mathrm{Stm}\ rs\ r)\ a$$

$$\mathsf{shift0} : ((a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ (r :: rs)\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \mathrm{Stm}\ (r :: rs)\ a \to (a \to \mathrm{Stm}\ rs\ r) \to \mathrm{Stm}\ rs\ r$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \mathrm{Stm}\ (a :: rs)\ a \to \mathrm{Stm}\ rs\ a$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{pure}$$

Representing Abstracted Control: Operators

$$\overline{\mathrm{Stm}} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\overline{\mathrm{Stm}}\ [\,]\ a = \underline{a}$$
$$\overline{\mathrm{Stm}}\ (r :: rs)\ a = \mathrm{Cps}\ (\overline{\mathrm{Stm}}\ rs\ r)\ \underline{a}$$

$$\mathsf{shift0} : ((\underline{a} \to \overline{\mathrm{Stm}}\ rs\ r) \to \overline{\mathrm{Stm}}\ rs\ r) \to \overline{\mathrm{Stm}}\ (r :: rs)\ a$$
$$\mathsf{shift0} = \mathsf{id}$$

$$\mathsf{run0} : \overline{\mathrm{Stm}}\ (r :: rs)\ a \to (\underline{a} \to \overline{\mathrm{Stm}}\ rs\ r) \to \overline{\mathrm{Stm}}\ rs\ r$$
$$\mathsf{run0} = \mathsf{id}$$

$$\mathsf{reset0} : \overline{\mathrm{Stm}}\ (a :: rs)\ a \to \overline{\mathrm{Stm}}\ rs\ a$$
$$\mathsf{reset0}\ m = \mathsf{run0}\ m\ \mathsf{pure}$$

Representing Abstracted Control: Monad

$$\mathrm{Stm} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\mathrm{Stm}\ [\,]\ a = a$$
$$\mathrm{Stm}\ (r :: rs)\ a = \mathrm{Cps}\ (\mathrm{Stm}\ rs\ r)\ a$$

$$\mathsf{pure} : a \to \mathrm{Stm}\ rs\ a$$
$$\mathsf{pure}_{r::rs}\ a = \lambda k \Rightarrow k\ a$$
$$\mathsf{pure}_{[\,]}\quad a = a$$

$$\mathsf{push} : (a \to \mathrm{Stm}\ (r :: rs)\ b) \to (b \to \mathrm{Stm}\ rs\ r) \to (a \to \mathrm{Stm}\ rs\ r)$$
$$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\mathsf{bind} : \mathrm{Stm}\ rs\ a \to (a \to \mathrm{Stm}\ rs\ b) \to \mathrm{Stm}\ rs\ b$$
$$\mathsf{bind}_{r::rs}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$$
$$\mathsf{bind}_{[\,]}\quad m\ f = f\ m$$

Representing Abstracted Control: Monad

$$\overline{\mathrm{Stm}} : \mathrm{List\ Type} \to \mathrm{Type} \to \mathrm{Type}$$
$$\overline{\mathrm{Stm}}\ [\,]\ a = \underline{a}$$
$$\overline{\mathrm{Stm}}\ (r :: rs)\ a = \mathrm{Cps}\ (\overline{\mathrm{Stm}}\ rs\ r)\ \underline{a}$$

$$\mathsf{pure} : \underline{a} \to \overline{\mathrm{Stm}}\ rs\ a$$
$$\mathsf{pure}_{r::rs}\ a = \lambda k \Rightarrow k\ a$$
$$\mathsf{pure}_{[\,]}\ \ a = a$$

$$\mathsf{push} : (\underline{a} \to \overline{\mathrm{Stm}}\ (r :: rs)\ b) \to (\underline{b} \to \overline{\mathrm{Stm}}\ rs\ r) \to (\underline{a} \to \overline{\mathrm{Stm}}\ rs\ r)$$
$$\mathsf{push}\ f\ k = \lambda a \Rightarrow f\ a\ k$$

$$\mathsf{bind} : \overline{\mathrm{Stm}}\ rs\ a \to (\underline{a} \to \overline{\mathrm{Stm}}\ rs\ b) \to \overline{\mathrm{Stm}}\ rs\ b$$
$$\mathsf{bind}_{r::rs}\ m\ f = \lambda k \Rightarrow m\ (\mathsf{push}\ f\ k)$$
$$\mathsf{bind}_{[\,]}\ \ m\ f = f\ m$$

Staged Statements: Reify and Reflect

mutual

reify : $\overline{\mathrm{Stm}}\ rs\ a \to \underline{\mathrm{Stm}\ rs\ a}$

reflect : $\underline{\mathrm{Stm}\ rs\ a} \to \overline{\mathrm{Stm}}\ rs\ a$

Staged Statements: Reify and Reflect

mutual

$\text{reify} : \overline{\mathrm{Stm}}\ rs\ a \rightarrow \underline{\mathrm{Stm}\ rs\ a}$

$\text{reify}_{[]}\quad m = m$

$\text{reify}_{q::qs}\ m = \underline{\lambda}\ \lambda k \Rightarrow \text{reify}\ (m\ (\lambda a \Rightarrow \text{reflect}\ \ (k\ \underline{@}\ a)))$

$\text{reflect} : \underline{\mathrm{Stm}\ rs\ a} \rightarrow \overline{\mathrm{Stm}}\ rs\ a$

$\text{reflect}_{[]}\quad m = m$

$\text{reflect}_{q::qs}\ m = \lambda k \Rightarrow \text{reflect}\ (m\ \underline{@}\ (\underline{\lambda}\ \lambda a \Rightarrow \text{reify}\ (k\ a)))$

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Conclusion

- Nicely fitting synthesis of:
  -

Conclusion

- Nicely fitting synthesis of:
  - Marek Materzok and Dariusz Biernacki. 2011. Sub *typing* Delimited Continuations. In Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). ACM, New York, NY, USA, 81–93.

Conclusion

- Nicely fitting synthesis of:
  - Marek Materzok and Dariusz Biernacki. 2011. Subtyping Delimited Continuations. In Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). ACM, New York, NY, USA, 81–93.
  - Olivier Danvy and Andrzej Filinski. 1992. *Representing* control: A study of the CPS transformation. Mathematical Structures in Computer Science 2, 4 (1992), 361–391.

Conclusion

- Nicely fitting synthesis of:
  - Marek Materzok and Dariusz Biernacki. 2011. Subtyping Delimited Continuations. In Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). ACM, New York, NY, USA, 81–93.
  - Olivier Danvy and Andrzej Filinski. 1992. Representing control: A study of the CPS transformation. Mathematical Structures in Computer Science 2, 4 (1992), 361–391.
  - Olivier Danvy and Andrzej Filinski. 1990. *Abstracting* Control. In Proceedings of the Conference on LISP and Functional Programming. ACM.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Conclusion

- Nicely fitting synthesis of:
  - Marek Materzok and Dariusz Biernacki. 2011. Subtyping Delimited Continuations. In Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). ACM, New York, NY, USA, 81–93.
  - Olivier Danvy and Andrzej Filinski. 1992. Representing control: A study of the CPS transformation. Mathematical Structures in Computer Science 2, 4 (1992), 361–391.
  - Olivier Danvy and Andrzej Filinski. 1990. Abstracting Control. In Proceedings of the Conference on LISP and Functional Programming. ACM.
- List of answer types are types of delimiters

Conclusion

- Nicely fitting synthesis of:
  - Marek Materzok and Dariusz Biernacki. 2011. Subtyping Delimited Continuations. In Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). ACM, New York, NY, USA, 81–93.
  - Olivier Danvy and Andrzej Filinski. 1992. Representing control: A study of the CPS transformation. Mathematical Structures in Computer Science 2, 4 (1992), 361–391.
  - Olivier Danvy and Andrzej Filinski. 1990. Abstracting Control. In Proceedings of the Conference on LISP and Functional Programming. ACM.
- List of answer types are types of delimiters
- Effect types guide CPS transform

Conclusion

- Nicely fitting synthesis of:
  - Marek Materzok and Dariusz Biernacki. 2011. Subtyping Delimited Continuations. In Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). ACM, New York, NY, USA, 81–93.
  - Olivier Danvy and Andrzej Filinski. 1992. Representing control: A study of the CPS transformation. Mathematical Structures in Computer Science 2, 4 (1992), 361–391.
  - Olivier Danvy and Andrzej Filinski. 1990. Abstracting Control. In Proceedings of the Conference on LISP and Functional Programming. ACM.
- List of answer types are types of delimiters
- Effect types guide CPS transform

Also in the paper:

- Avoiding Eta-Redexes
- Branching and Recursion